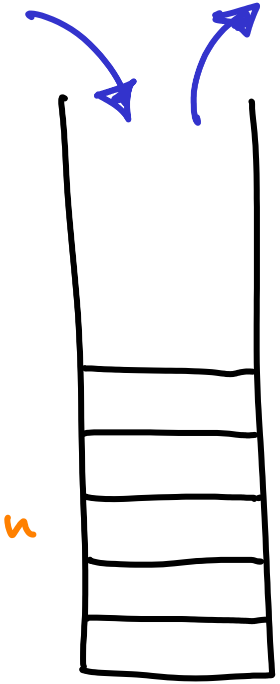
Stack operations: push pop multipop(k)

Cost = 1 Cost = 1 Cost = k



What will n operations cost?

Worst case: $n \cdot (\text{max individual op}) = n \cdot k = k$ per operation

Amortized $= O(1)$ per operation

Intuitive: each element can be pushed precisely once
but also only popped or multipopped once

Contribution per element $\leq 2$ / #elements $\leq n$ / Total cost $= O(n)$

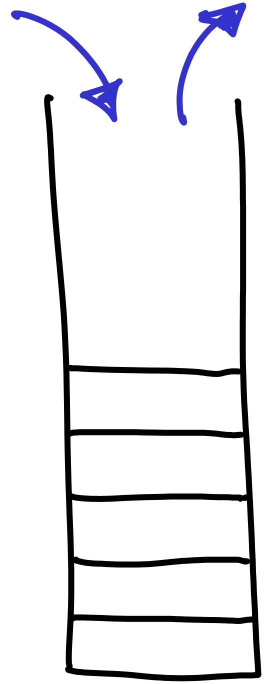
# AMORTIZATION

Applies to some problems that involve many operations.

If worst case time of operation $k$ is $O(f(k))$,
try to show that $n$ operations cost $o(n \cdot f(n))$

Stack operations:

| | push | pop | multipop(k) |
|---|---|---|---|
| | Cost = 1 | Cost = 1 | Cost = k |
| | Pretend 2 | use savings | |
| Amortized cost: | 2 | 0 | 0 |



stack

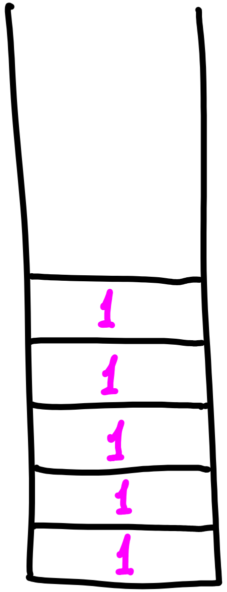
## Accounting method

- Pretend some cheap operation type costs more
- Save the difference, use it later

True cost ≤ n · (worst amortized cost)

= n · 2

BANK →

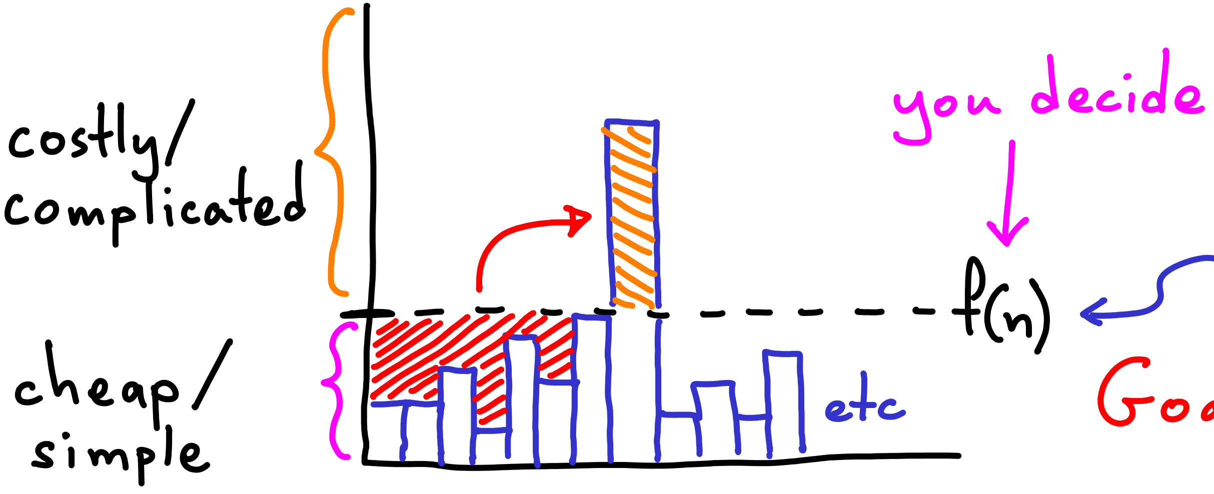


# Accounting : saving for a rainy day

Pretend "simple" operations cost more than they do. Ideally $\Theta$(real cost)

↳ "save" the difference → "spend" what you saved up.

"Complicated/costly" operations : pretend they cost less; pay excess via savings




Rule : Never spend more than what you saved.

Amortized cost : $n \cdot f(n) \geq$ true cost

Goal : exaggerate / save as little as possible. } minimize $f(n)$

# POTENTIAL METHOD aka Physicist's method

Start with data structure $D_0$

Operation $i$ : $D_{i-1} \to D_i$ cost : $c_i$

Potential function $\Phi_i$ maps $D_i \to \mathbb{R}$ : potential value.

$\Phi_0 = 0$ $\Phi_i \geqslant 0$ $\Rightarrow$ 2 conditions that help.

Let $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$
$= c_i + \Delta\Phi_i$

$\left\{ \begin{array}{l} \text{If } \Delta\Phi_i > 0, \ \hat{c}_i > c_i : \text{storing potential ... "work" in } D_i \\ \Delta\Phi_i < 0, \ \hat{c}_i < c_i : \text{release work.} \end{array} \right.$

$\hat{c}_i = c_i + \Delta\Phi_i \Rightarrow \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$= \underbrace{\Phi_n}_{\geq 0} - \underbrace{\Phi_0}_{0} + \sum_1^n c_i \geq \sum_1^n c_i$

so we know that the amortized cost will not underestimate real cost.

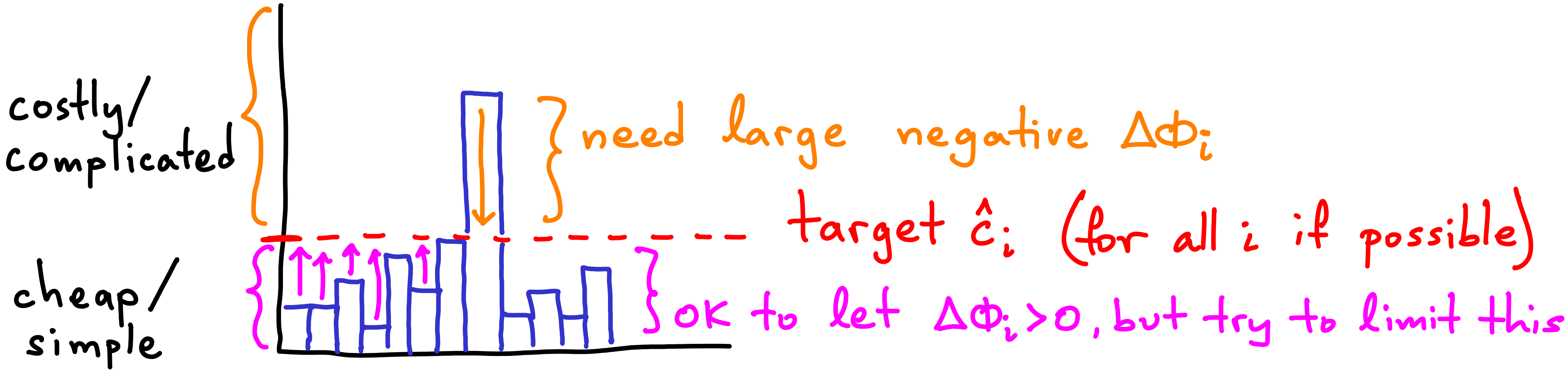
now figure out worst case for any individual $\hat{c}_i$

↳ Ideally this will give a good (and easy) bound for total cost

$$\sum c_i \leq \sum \hat{c}_i \leq n \cdot \max \hat{c}_i$$

# How this works

- (Subjectively) define what a complicated / costly operation ($c_i$) is.
- Find something that changes a lot in the data structure in such cases.
  - ↪ Quantify this change as $\Delta\Phi_i$ : let it "kill" $c_i$ : $\hat{c}_i = c_i + \Delta\Phi_i$
  - ↪ Invent your $\Phi$ accordingly ↪ obtain low $\hat{c}_i$
  - ↪ Make sure $\Delta\Phi_i$ doesn't add much to $c_i$, when $c_i$ is cheap.



# Potential method

Let $\Phi$ = # elements in stack

Let $\hat{c}_i = c + \Delta\Phi_i$

if $\geq 0$ then $\sum_{i=1}^{n} c \leq \sum_{i=1}^{n} \hat{c}_i$

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c + \sum_{i=1}^{n} \Delta\Phi_i = \sum_{i=1}^{n} c + \overbrace{\Phi_n - \Phi_0}^{\text{if } \geq 0}$$

$\geq 0$

$\Phi_0 = 0$ if stack starts empty

---

Stack operations: push pop multipop(k)

| | push | pop | multipop(k) |
|---|---|---|---|
| | Cost = 1 | Cost = 1 | Cost = k |
| $\Delta\Phi$ | +1 | -1 | -k |
| $\hat{c}_i =$ | 1 + 1 | 1 − 1 | k − k |

$$\sum_{i=1}^{n} \hat{c}_i \leq n \cdot \max \hat{c}_i = n \cdot 2 \rightarrow \sum_{i=1}^{n} c \leq 2n$$

Incrementing a k-bit counter: cost of n increments? (assume max = 111111...)

$(< 2^k)$

cost = # bits flipped. $0 \to 1$, $1 \to 0$

$\Phi$: # leading 1's

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

$\Phi = 0$

$\hat{c} = 1 + (1 - 0) = 2$

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

$\Phi = 1$

$\hat{c} = 2 + (0 - 1) = 1$

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

$\Phi = 0$

$\hat{c} = 1 + (10 - 0) = 11$ → allowed small c to grow a lot

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

$\Phi = 10$

$\hat{c} = 11 + (0 - 10) = 1$ → killed large c

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$\Phi = 0$

Incrementing a k-bit counter: cost of $n$ $(<2^k)$ increments? (assume max = 11111...)

cost = # bits flipped. $0 \to 1$, $1 \to 0$

| counter | $\Phi$: # leading 1's | $\Phi$ = total # 1's |
|---|---|---|
| 1 0 0 1 1 1 1 1 1 1 1 0 0 | $\Phi = 0$ | $\Phi = 9$ |
| | $\hat{c} = 1 + (1-0) = 2$ | $\hat{c} = 1 + (10-9) = 2$ |
| 1 0 0 1 1 1 1 1 1 1 1 0 1 | $\Phi = 1$ | $\Phi = 10$ |
| | $\hat{c} = 2 + (0-1) = 1$ | $\hat{c} = 2 + (10-10) = 2$ |
| 1 0 0 1 1 1 1 1 1 1 1 1 0 | $\Phi = 0$ | $\Phi = 10$ |
| | $\hat{c} = 1 + (10-0) = 11$ ☹ | $\hat{c} = 1 + (11-10) = 2$ |
| 1 0 0 1 1 1 1 1 1 1 1 1 1 | $\Phi = 10$ | $\Phi = 11$ |
| | $\hat{c} = 11 + (0-10) = 1$ | $\hat{c} = 11 + (2-11) = 2$ ☺ |
| 1 0 1 0 0 0 0 0 0 0 0 0 0 | $\Phi = 0$ | $\Phi = 2$ |

$\Phi$: #leading 1's $\rightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to $k-1$)

$\sum \hat{c} \le n \cdot \max(\hat{c}) = n(k-1)$

↳ $\le k-1$ per increment

$\Phi$ = total #1's

First 0 at position $i+1$ $\rightarrow$ $\left.\begin{array}{l} c = i+1 \\ \Delta\Phi = 1-i \end{array}\right\}$ $\hat{c}$: 2 (always)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| | | $i+1$ | ...... | 3 | | 2 | 1 |

$\sum \hat{c} \le n \cdot \max(\hat{c}) = 2n$

But is $\sum c \le \sum \hat{c}$ ? $\left(\sum \hat{c} = \sum c + \Phi_n - \Phi_0\right)$

$\Phi_n \ge 0$ but $\Phi_0$ not necessarily 0, so $\Phi_n - \Phi_0$ could be negative ✗

↳ $= 0$ if we start count up from zero ✓

$\Phi \le k \rightarrow \Phi_n - \Phi_0 \ge -k \rightarrow \sum \hat{c} \ge \sum c - k \rightarrow \sum c \le k + \sum \hat{c} \rightarrow$

$\rightarrow \sum c \le k + 2n \rightarrow$ if $n \ge k$, $\sum c \le 3n$

# Incrementing a k-bit counter — ACCOUNTING

Every $0 \to 1$ will cost $\hat{c} = 2$ (instead of $c = 1$)

Use the extra 1 to pay for $1 \to 0$ for the same bit (later)

↳ $1 \to 0$ will cost $\hat{c} = 0$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| | | $i+1$ | . . . | . . . | 3 | 2 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

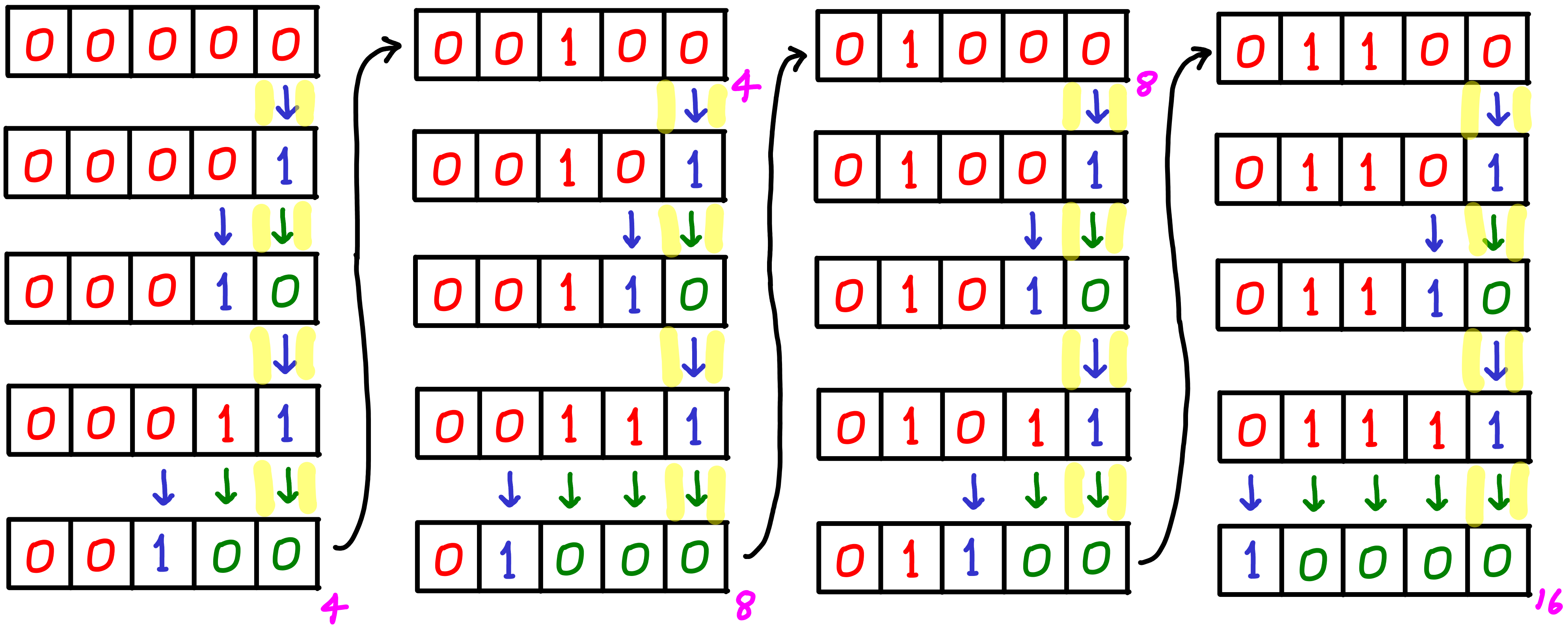
BANK

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | | | 1 | 1 | 1 | 1 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | | 1 | | | | | |

## Aggregate method: Just count everything

* relies on full understanding of operation types, frequencies, costs
* not always possible



Incrementing a k-bit counter – aggregate: cost = $\underbrace{\boxed{\phantom{n}}\boxed{n/2^i}\cdots\boxed{n/4}\boxed{n/2}\boxed{n}}_{2n}$

(n times)

index $i$ gets flipped every $2^i$ iterations: total $\displaystyle\sum_{i=0}^{n} \frac{n}{2^i}$